Starviewer and its comparison with other open-source DICOM viewers using a novel hierarchical evaluation framework

Marc Ruiz[a], Adrià Julià[a], Imma Boada[a]

[a]*Graphics and Imaging Laboratory (GILAB)*
Universitat de Girona, Edifici P-IV
17001 Girona, Catalonia

Abstract

Methods: The aim of the paper is twofold. First, we present Starviewer, a DICOM viewer developed in C++ with a core component built on top of open-source libraries. The viewer supports extensions that implement functionalities and front-ends for specific use cases. Second, we propose an adaptable evaluation framework based on a set of criteria weighted according to user needs. The framework can consider different user profiles and allow criteria to be decomposed in subcriteria and grouped in more general categories making a multi-level hierarchical structure that can be analysed at different levels of detail to make scores interpretation more comprehensible.

Results: Different examples to illustrate Starviewer functionalities and its extensions are presented. In addition, the proposed evaluation framework is used to compare Starviewer with four open-source viewers regarding their functionalities for daily clinical practice. In a range from 0 to 10, the final scores are: Horos (7.7), Starviewer (6.2), Weasis (6.0), Ginkgo CADx (4.1), and medInria (3.8).

Conclusions: Starviewer provides basic and advanced features for daily image diagnosis needs as well as a modular design that enables the development of custom extensions. The evaluation framework is useful to understand and prioritize new development goals, and can be easily adapted to express different needs by altering the

*Contacting author

Email addresses: `marc.ruiz@udg.edu` (Marc Ruiz), `adria.julia@udg.edu` (Adrià Julià), `imma.boada@udg.edu` (Imma Boada*)

weights. Moreover, it can be used as a complement to maturity models and quality evaluation frameworks.
Keywords: DICOM viewer, open-source, software comparison, medical imaging

1. Introduction

DICOM, that stands for Digital Imaging and Communications in Medicine, is the leading standard for image data management in medical applications and is used to capture, exchange, and archive image data in Picture Archiving and Communication Systems (PACS) [1]. PACS can be seen as a centralized repository of all medical images. To diagnose, physicians require a DICOM viewer with features to connect with the PACS in order to retrieve and store images and with other functionalities to visualize, explore and analyse the information represented in these images [2].

There is a wide variety of privative and open-source DICOM viewers. Focusing on the latter, their functionalities range from simple 2D or 3D visualizations to more advanced techniques and image processing tools to measure the volume of lesions or combine information from different image modalities, among others. Recently, Valeri et al. [3] evaluated the most representative viewers for GNU/Linux, Windows, and macOS; beeing the best viewers 3D Slicer [4, 5], medInria [6], MITK Workbench (MITK 3M3 before 2009) [7, 8], VolView (currently deprecated) [9], VR Render (currently deprecated) [10], and OsiriX [11, 12]. More recently, Haak et al. [13] did an evaluation centering not only on viewing functionalities, but also on platforms and interfaces. MIPAV [14, 15] and Weasis [16] were identified as superior open-source tools. In the macOS community, the most popular open-source viewer was OsiriX [11] which is currently a proprietary solution. However, the last open-source version of OsiriX was forked into several projects by the community, being Horos [17] the most popular one.

The offer of open-source software is continuously growing in all the areas of research and practice and also in the medical one. Selecting the product that better fits the health environment and more specifically user needs becomes a complex task. In this context, under the premise that the quality of a software system is largely determined by the quality of the software process used to build it, capability/maturity models have been proposed. These can be represented in a two-dimensional space where a first dimension represents what is done and a second one of how well it is done, i.e. the capability/maturity dimension. These models have a set of levels ranging from the lowest, where no conception of maturity is defined, to the highest, representing total capability maturity. To determine the level of a software quality, attributes such as functionality, reliability, usability, efficiency, maintainability, and portability are considered. In the healthcare domain, several maturity models have been proposed but they are still at an early stage of development [18, 19, 20]. For a review on this topic see [21] and for a comparison of maturity models on open-source software see [22]. Unfortunately, no consensus on the aspects or the models that have to be considered for an evaluation in a specific area exists. This leads, in some cases, the selection of a DICOM viewer to a process based on recommendation and past experiences [23, 24].

A key point of maturity models relies on the selection of the parameters to be evaluated and the grading system used to evaluate them. In the case of open-source DICOM viewers, different parameters and grading systems have been proposed to compare them. On the one hand, there are authors that consider the most important features to be installation facilities, support, and documentation [25], while others consider the provided working functionalities more important [26, 3, 13]. On the other hand, grading approaches range from the simplest, where yes/no classification are considered [13, 3] to the most elaborated grading scales [26]. With the aim of satisfying as many audiences as possible, we propose a fully customizable evaluation framework with no limits on evaluated criteria and with the possibility to modify the grading strategy. The framework uses a hierarchical structure to maintain information grouped in audiences, groups, criteria, and subcriteria. Weights are assigned for each node and are distributed across the sibling nodes for each level in the hierarchy. Scores range from 0 to 1 and are computed as a weighted sum which is propagated to the upper nodes providing intermediate weighted scores for each level in the hierarchy. For more details about the calculation refer to Fig. 2, the actual supporting material spreadsheet, and the Appendix B. In this way, a rapid evaluation of the software can be obtained with more or less detail depending on the user interests. The proposed framework cannot be a substitute of a maturity model but a complement to determine the values of parameters required by the models.

Although our framework mainly focuses on technological aspects, the overall evaluation of free/open-source projects requires the consideration of other aspects such as production, distribution and support. Different models to evaluate these factors have been proposed. Kamseu and Habra [27] presented a three dimensional model where the development process, the community, and the project are evaluated. The *Qualification and Selection of Open Source Software* (QSoS) applies four independent and iterative steps aimed at defining, evaluating, qualifying, and selecting open-source solutions based on software support and technology [28]. The *Quality Platform for Open Source Software (QualiPSo)*, includes an evaluation framework regarding the trustworthiness of open-source projects [29, 30]. Sung et al. adapted the ISO/IEC 9126 standard to free/open-source projects focusing on the quality of the products [31]. The *Evaluation Framework for Free/Open source projects* (EFFORT), is a framework that, once customized for a specific context, supports the evaluation of product quality, attractiveness, and community trustworthiness [32, 33]. For more details on these methods see [33]. Note that unlike described methods our approach focuses on functionalities, audiences and technological aspects of medical imaging software but not on production, distribution and support. Therefore, we consider it as a suitable complement to the described methods and not a substitute.

Besides this introduction, the paper has been structured as follows. In Section 2, the architecture of Starviewer DICOM viewer is presented, as well as the proposed evaluation framework and the evaluation procedure that has been applied to compare different open-source radiological viewers with the proposed one. In Section 3, the main functionalities of Starviewer platform are illustrated and also the results obtained from the comparison of selected open-source radiological viewers. The obtained results are discussed in Section 4.

2. Materials and methods

2.1. Starviewer architecture

Starviewer was conceived as an integrable open-source solution for DICOM-based diagnosis. To design it we considered fundamental to support basic and advanced functionalities and also the possibility to extend the viewer in order to assist specific workflows required by experts. We designed the three-level architecture illustrated in Fig. 1 and described below.

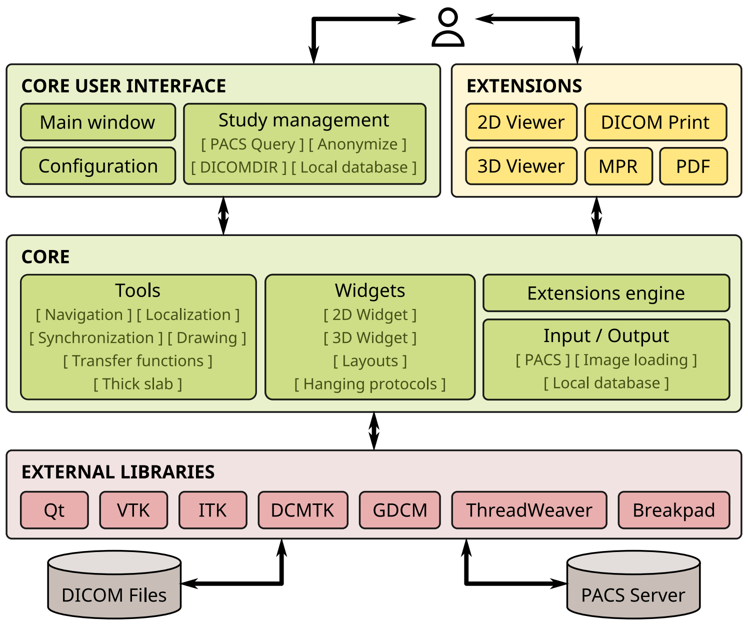


Figure 1: Starviewer architecture block diagram.

2.1.1. *External libraries*

The first level of the Starviewer architecture contains the open-source libraries used by the platform. In particular, Qt [34] is used as the development framework, and VTK [35] and ITK [36] for image representation, processing, visualization, interaction, and rendering. DCMTK [37] is used to communicate with the PACS and as the primary choice when reading DICOM files. GDCM [38] is used by the anonymization feature and as a surrogate when reading unsupported DCMTK image codecs such as JPEG2000 (which is not included in the free version of DCMTK). ThreadWeaver [39], which simplifies the implementation of multi-threaded asynchronous algorithms using a thread pool, is used for image loading, PACS queries, downloads, and uploads. Finally, Breakpad [40] is a library used to create crash reports with memory dumps that are stored locally and then sent to our servers for further analysis.

2.1.2. Core

The second level of the Starviewer architecture has two main components, the core and the core user interface. The Starviewer core is made of algorithms, common user interfaces, and general purpose modules that conform the structure which can later be leveraged by the extensions. It has four main parts.

First, *input/output* modules and user interfaces consisting of: (i) the *local database*, (ii) the *PACS connection* module, and (iii) the *image loading* module.

Second, the *extensions engine*, that provides some classes and specifies a structured methodology to extend the program functionalities in order to satisfy new user requirements.

Third, *widgets* that can be divided in: (i) *2D widget*, focused on visualizing plain images, slices from volumes, and fusions between series; (ii) *3D widget*, targeting the visualization and rendering of volumetric datasets; (iii) *layouts*, whose task is to arrange a set of viewer widgets; and (iv) *hanging protocols*, which are a set of user-definable templates aware of the study context that have rules to decide when a hanging protocol is applicable, define a layout with custom-sized viewers, and have sets of rules to determine which contents shall be placed in each viewer.

Fourth, *tools*, a wide collection of components that can be associated with the basic 2D and 3D viewer widgets to add interaction and extend their functionality. They are used to implement or reuse extension-specific features and are divided in six categories: (i) *navigation* tools, (ii) *transfer function* tools, (iii) *thick slab* tool, (iv) *localization* tools, (v) *drawing* tools, and (vi) *synchronization* tools.

2.1.3. Core user interface

Besides core, in the second level of the architecture there is also the core user interface. This exposes common features that are not implemented by extensions but are required by them. Its main elements are: (i) the *main window*, (ii) the *configuration* *window*, and and (iii) the *study management* window.

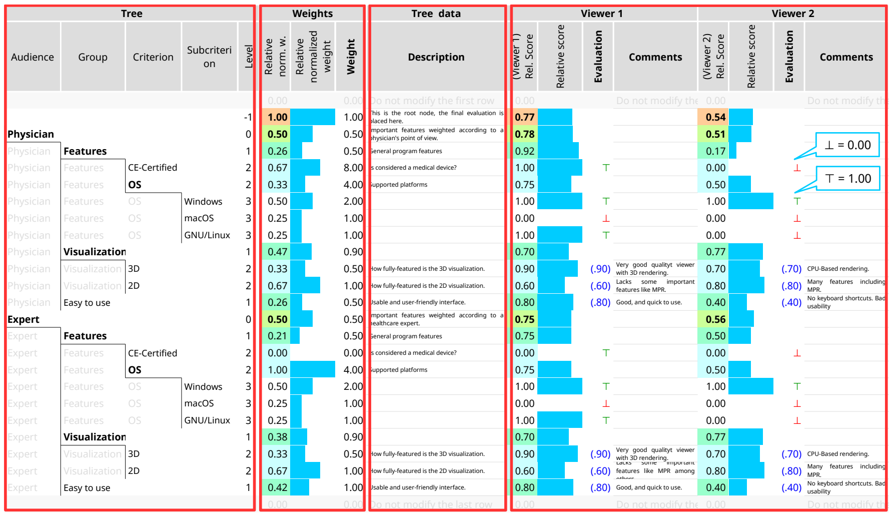
2.1.4. Extensions

In the last level of the architecture, there are the extensions. These make use of the underlying tools, widgets, and the Starviewer platform in general to develop specialized tools for specific use cases with custom user interfaces. An extension is implemented when a user requires a set of functionalities to deal with a determined protocol or situation. Although the functionalities of extensions are different, for the sake of simplicity for the user, they usually follow a standard layout pattern with two distinct regions: a *toolbar*, with shortcuts to main functionalities, camera options, and tools; and a *views area* containing a layout of viewers, which can be 2D or 3D, or other specialized widgets to display information. A detailed description of how to create an extension is given in Appendix A.

2.2. The evaluation framework

Usually, software feature comparisons consist in gathering user needs, representing them as a set of criteria, and grading them following a basic methodology [3, 13] or a more complex one which varies the requirements, adapts the weights, and uses fine-grained criteria [41, 42]. Our evaluation framework expands on some of these ideas using a very fine-grained tree of hierarchically organized criteria where related features are grouped and scored. Every node of the tree is assigned a weight according to its relative importance in its context (sibling nodes); weights are automatically normalized at each context (sibling nodes) to sum 1. Then, each leaf node is evaluated and given an score in the range $[0, 1]$ being the parent node score the weighted sum of them.

The proposed framework is presented in Fig. 2 where its main blocks have been highlighted. By default the evaluator has a template where the four first columns (see block (a)) represent the levels of the tree structure, where from left to right, each column represents: (i) *audiences*; (ii) *groups*, a high-level grouping of criteria not meant to be directly evaluated; (iii) *criteria*, specific items to evaluate; and (iv) *subcriteria*, present when a criterion is finely evaluated. The evaluator may alter the tree by adding, removing, or rearranging criteria, and adapting the weights of the rows to express his or her needs. Weights are presented in block (b). The user enters the weights in the last column and they are automatically normalized and shown both numerically and graphically in the first and second columns. In addition, there is a column to enter a textual description of the criteria (see block (c)). Finally, in block (d), the software products (in our case DICOM viewers) to be compared are placed. The evaluator enters the score in the "evaluation" column for each leaf node (criterion or subcriterion), and the relative score and graph bar are automatically computed. If a new software solution has to be evaluated, a new group of columns must be appended at the end.



Tree					Weights			Tree data	Viewer 1				Viewer 2			
Audience	Group	Criterion	Subcriterion	Level	Relative norm. w.	Relative normalized weight	**Weight**	**Description**	(Viewer 1) Rel. Score	Relative score	**Evaluation**	**Comments**	(Viewer 2) Rel. Score	Relative score	**Evaluation**	**Comments**
					0.00		0.00	Do not modify the first row	0.00			Do not modify the	0.00			Do not modify th
				-1	**1.00**		1.00	This is the root node, the final evaluation is placed here.	**0.77**				**0.54**			
Physician				0	**0.50**		0.50	mportant features weighted according to a physician's point of view.	**0.78**				**0.51**			
Physician	**Features**			1	0.26		0.50	General program features	0.92				0.17			
Physician	Features	CE-Certified		2	0.67		8.00	s considered a medical device?	1.00		⊤		0.00		⊥	
Physician	Features	**OS**		2	0.33		4.00	Supported platforms	0.75				0.50			
Physician	Features	OS	Windows	3	0.50		2.00		1.00		⊤		1.00		⊤	
Physician	Features	OS	macOS	3	0.25		1.00		0.00		⊥		0.00		⊥	
Physician	Features	OS	GNU/Linux	3	0.25		1.00		1.00		⊤		0.00		⊥	
Physician	**Visualization**			1	0.47		0.90		0.70				0.77			
Physician	Visualization	3D		2	0.33		0.50	How fully-featured is the 3D visualization.	0.90		(.90)	Very good qualityt viewer with 3D rendering.	0.70		(.70)	CPU-Based rendering.
Physician	Visualization	2D		2	0.67		1.00	How fully-featured is the 2D visualization.	0.60		(.60)	Lacks some important features like MPR.	0.80		(.80)	Many features including MPR.
Physician	Easy to use			1	0.26		0.50	Usable and user-friendly interface.	0.80		(.80)	Good, and quick to use.	0.40		(.40)	No keyboard shortcuts. Bad usability
Expert				0	**0.50**		0.50	mportant features weighted according to a healthcare expert.	**0.75**				**0.56**			
Expert	**Features**			1	0.21		0.50	General program features	0.75				0.50			
Expert	Features	CE-Certified		2	0.00		0.00	s considered a medical device?	0.00		⊤		0.00		⊥	
Expert	Features	**OS**		2	1.00		4.00	Supported platforms	0.75				0.50			
Expert	Features	OS	Windows	3	0.50		2.00		1.00		⊤		1.00		⊤	
Expert	Features	OS	macOS	3	0.25		1.00		0.00		⊥		0.00		⊥	
Expert	Features	OS	GNU/Linux	3	0.25		1.00		1.00		⊤		0.00		⊥	
Expert	**Visualization**			1	0.38		0.90		0.70				0.77			
Expert	Visualization	3D		2	0.33		0.50	How fully-featured is the 3D visualization.	0.90		(.90)	Very good qualityt viewer with 3D rendering.	0.70		(.70)	CPU-Based rendering.
Expert	Visualization	2D		2	0.67		1.00	How fully-featured is the 2D visualization.	0.60		(.60)	Lacks some important features like MPR among others.	0.80		(.80)	Many features including MPR.
Expert	Easy to use			1	0.42		1.00	Usable and user-friendly interface.	0.80		(.80)	Good, and quick to use.	0.40		(.40)	No keyboard shortcuts. Bad usability
					0.00		0.00	Do not modify the last row	0.00			Do not modify the	0.00			Do not modify th

⊥ = 0.00

⊤ = 1.00

(a) (b) (c) (d)

Figure 2: The main blocks of the proposed evaluation framework where several auxiliary columns are hidden for the sake of readability. (a) Audiences, groups, criteria, and subcriteria are organized hierarchically; the tree structure is drawn automatically from the entered data. (b) The user enters weights in the bold column for each row in the tree; relative normalized weights and graph bars are automatically computed. (c) Description for each node of the tree. (d) Viewer scores: for each viewer a set of columns is added; the user enters the score in the "evaluation" column for each leaf node; relative scores and graph bars are automatically computed.

2.3. The evaluation procedure

2.3.1. Evaluation framework definition

To fill the framework with criteria and weights specific to DICOM viewers, we restricted our study to functional requirements which where compiled from: (i) clinical experts feedback; (ii) our knowledge with the development of Starviewer; (iii) existing literature [3, 13, 26]; and (iv) a collection of the analysed viewers features. Although they are equally important, non-functional requirements such as security, privacy, availability, platform usability, or efficiency were not evaluated. However, they can be easily integrated in the framework. To select functional requirements we evaluated state of the art DICOM viewers (see Section 2.3.2) and we elaborated a list with their functionalities. We grouped these functionalities in seven groups: (i) *technical* features, (ii) *archive*, (iii) *workflow*, (iv) *visualization* and interaction with a volume or set of images, (v) *tools*, (vi) *other modalities*, and (vii) *processing* features. Then, we collected this information in a survey and asked radiologists to grade them according to importance in a range from 0 to 10. The survey includes a question asking the number of years each respondent has been diagnosing from medical images and their area of diagnosis. The survey was sent to 25 radiologists of different centers of Catalan Hospitals. From their answers, we created a first list of criteria and weights which were obtained from the mean grades. List items were transformed to the evaluation framework criteria. In addition, from the collected information we decided to consider three broad types of audiences: (i) *general*, which reflects the common needs of a physician, not exclusively focused on 2D or 3D; (ii) *2D-based* audience, that works with plain images like ultrasound, mammography, and X-ray; and (iii) *3D-based* audience whose datasets are mainly volumetric images from computed tomography (CT) and magnetic resonance (MRI) as well as more specific nuclear medicine modalities like positron emission tomography (PET) or single-photon emission computed tomography (SPECT). In the evaluation, these three audiences have the same criteria and evaluations, but their weights have been tuned to more accurately reflect their particular audience needs.

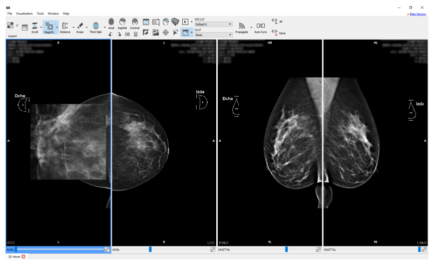
The filled framework was supervised by a group of experienced radiologists via personal interviews. They were asked to pinpoint missing criteria and check and fine-tune the weights. From this process we obtain the filled evaluation framework presented in Section 3.2.

2.3.2. Selection of viewers to compare

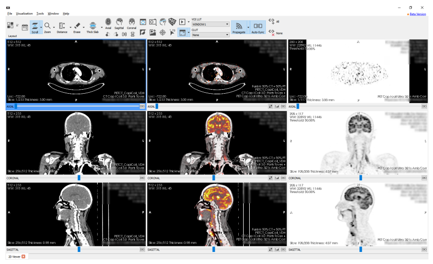
To select the open-source software candidates, a search focused on viewers for daily clinical practice was carried out. The search was conducted using: general purpose search engines like DuckDuckGo and Google; Debian, Fedora, and Ubuntu package archives; specialized websites like IDoImaging.com [43] and AlternativeTo.net [44]; and popular code forges like GitHub [45], Bitbucket [46], GitLab [47], and SourceForge [48].

A list of 54 candidates was produced from which viewers being unmaintained or not achieving the minimum requirements for daily clinical practice were discarded. This reduces the list to eleven pieces of software (Aliza [49], Amide [50], Ginkgo CADx [51], Horos [17], InVesalius [52], medInria [6], MIPAV [15], MITK [8], Slicer [5], Starviewer, and Weasis [16]) that were installed in a multi-platform testing environment with a PACS server for further analysis. After a first screening, six of them (Aliza, Amide, InVesalius, MIPAV, MITK, and Slicer), all excellent tools, were discarded because they were more research and processing focused. This leads to the final list with Ginkgo CADx, Horos, medInria, Starviewer, and Weasis.

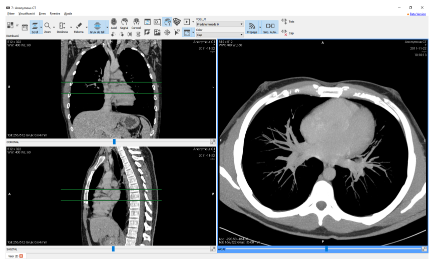
Once the viewers to compare were selected and the evaluation framework was filled with criteria and weights, a group of three experts with experience on radiological viewers but not involved in the development of Starviewer carried out the comparison. The evaluation was performed taking in consideration the description of each item combined with the need to provide an individual justification for the more qualitative items. Each expert carried the evaluation individually; many items could be evaluated in a boolean fashion, however when this could not be done a grading between 0 and 1 was used instead. In a second stage the grades were shared in order to discuss a final consensus.



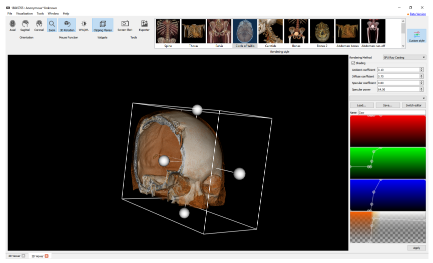
(a)



(b)



(c)



(d)

Figure 3: Screenshots showing several features of Starviewer. (a) 2D viewer with the magnifying glass tool. (b) 2D viewer visualizing a PET-CT with axial, coronal and sagittal reconstructions in the rows combined with the CT, PET-CT fusion and PET modalities in the columns. (c) 2D viewer showing the reference lines tool with thick slab enabled; the green lines in the coronal and sagittal viewers show the location of the axial view and its thickness. (d) 3D viewer with clipping planes enabled and a sidebar to modify the current transfer function and illumination parameters.

3. Results

3.1. Starviewer as a platform

Starviewer is developed in C++11 with support from several open-source libraries: Qt, DCMTK, ITK, VTK, GDCM, ThreadWeaver, and Breakpad. It is supported in Windows, macOS, and GNU/Linux operating systems. The source code is available at `http://www.starviewer.org` and `https://github.com/starviewer-medical/starviewer` licensed under GPLv3+ terms.

Starviewer core is extended with five stable extensions: *2D viewer* (with support for fusion), *3D viewer*, *MPR* (multi-planar reconstruction), *DICOM Print*, and *PDF*. To illustrate their functionalities, some screenshots are shown in Fig. 3.

Apart from the stable extensions, there are some experimental ones that are examples of research focused extensions: (i) *diffusion perfusion segmentation*; (ii) *edema segmentation*, which calculates the volume of automatically detected lesions in the brain (edemas and hematomas); and (iii) *rectum segmentation*. These extensions are not included in regular compiled binaries, but are available in the source code.

3.2. Evaluation

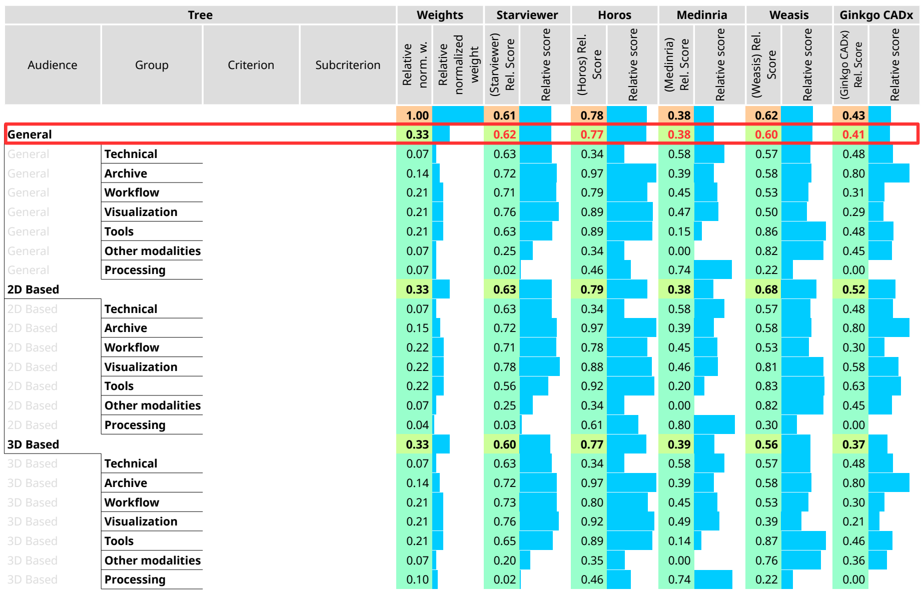
The results have to be interpreted as the capability of a viewer to address specific audience needs. A weighted and fine grained bottom-up evaluation of the features is performed in order to reduce the bias. Depending on the audience some needs may be more important than others. In the presented evaluation we expressed three broad types of audience that try to mimic the daily clinical practice needs.

The proposed evaluation framework has been implemented in a macro-free LibreOffice spreadsheet that automatically draws the tree, displays warnings, and has built-in help, among other advanced features. It is available as supporting material, and in `https://github.com/starviewer-medical/dicom-viewers-comparison`.

In Fig. 4 the results of the comparison of Starviewer with Ginkgo CADx, Horos, medInria, and Weasis using the proposed evaluation framework is presented. For a better comprehension, these results are shown only up to group level where each value is obtained from a weighted sum of the scores present at lower levels. The full evaluation can be seen in the spreadsheet in the supporting material, as illustrated in Fig. 5, where the “tools” group and the “annotations” criterion have been expanded.

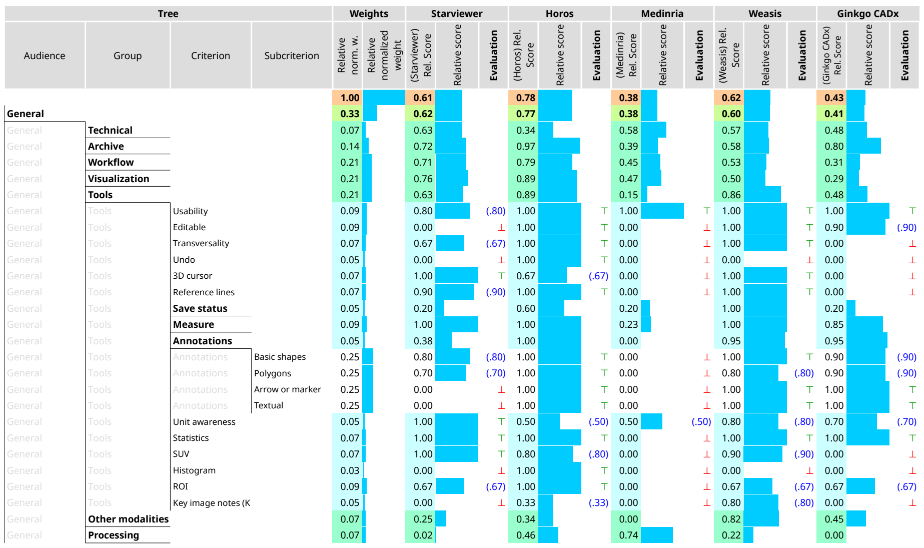
From Fig. 4 we can observe that for a general audience the best score is obtained by Horos with 0.78, while Weasis and Starviewer have a similar score, 0.62 and 0.61, respectively. The worst positions are for Gingko CADx, and medInria, with 0.43 and 0.38, respectively. Focusing on 2D users, Ginkgo CADx (0.52) and Weasis (0.68) significantly increase their scores relative to the general audience. The other viewers show a slight increase, relegating Starviewer (0.63) to the third position. Finally, focusing on 3D users, rankings stay the same as in the general audience, but more 2D oriented viewers, Ginkgo CADx (0.37) and Weasis (0.56), decrease their scores significantly.

In addition, Fig. 6 shows the summarized results for the general audience in a



Tree				Weights		Starviewer		Horos		Medinria		Weasis		Ginkgo CADx	
Audience	Group	Criterion	Subcriterion	Relative norm. w.	Relative normalized weight	(Starviewer) Rel. Score	Relative score	(Horos) Rel. Score	Relative score	(Medinria) Rel. Score	Relative score	(Weasis) Rel. Score	Relative score	(Ginkgo CADx) Rel. Score	Relative score
				1.00		**0.61**		**0.78**		**0.38**		**0.62**		**0.43**	
General				**0.33**		0.62		0.77		0.38		0.60		0.41	
General	**Technical**			0.07		0.63		0.34		0.58		0.57		0.48	
General	**Archive**			0.14		0.72		0.97		0.39		0.58		0.80	
General	**Workflow**			0.21		0.71		0.79		0.45		0.53		0.31	
General	**Visualization**			0.21		0.76		0.89		0.47		0.50		0.29	
General	**Tools**			0.21		0.63		0.89		0.15		0.86		0.48	
General	**Other modalities**			0.07		0.25		0.34		0.00		0.82		0.45	
General	**Processing**			0.07		0.02		0.46		0.74		0.22		0.00	
2D Based				**0.33**		**0.63**		**0.79**		**0.38**		**0.68**		**0.52**	
2D Based	**Technical**			0.07		0.63		0.34		0.58		0.57		0.48	
2D Based	**Archive**			0.15		0.72		0.97		0.39		0.58		0.80	
2D Based	**Workflow**			0.22		0.71		0.78		0.45		0.53		0.30	
2D Based	**Visualization**			0.22		0.78		0.88		0.46		0.81		0.58	
2D Based	**Tools**			0.22		0.56		0.92		0.20		0.83		0.63	
2D Based	**Other modalities**			0.07		0.25		0.34		0.00		0.82		0.45	
2D Based	**Processing**			0.04		0.03		0.61		0.80		0.30		0.00	
3D Based				**0.33**		**0.60**		**0.77**		**0.39**		**0.56**		**0.37**	
3D Based	**Technical**			0.07		0.63		0.34		0.58		0.57		0.48	
3D Based	**Archive**			0.14		0.72		0.97		0.39		0.58		0.80	
3D Based	**Workflow**			0.21		0.73		0.80		0.45		0.53		0.30	
3D Based	**Visualization**			0.21		0.76		0.92		0.49		0.39		0.21	
3D Based	**Tools**			0.21		0.65		0.89		0.14		0.87		0.46	
3D Based	**Other modalities**			0.07		0.20		0.35		0.00		0.76		0.36	
3D Based	**Processing**			0.10		0.02		0.46		0.74		0.22		0.00	

Figure 4: Summarized results of the comparison of Starviewer with Horos, medInria, Weasis, and Ginkgo CADx considering three different audiences (general, 2D-based, and 3D-based), and criteria grouped in seven categories. Criteria and subcriteria are hidden for comprehensibility. The final scores for the general audience (highlighted in red), are the ones we consider more relevant as they are in the most transversally weighted audience.



Tree				Weights		Starviewer			Horos			Medinria			Weasis			Ginkgo CADx		
Audience	Group	Criterion	Subcriterion	Relative norm. w.	Relative normalized weight	(Starviewer) Rel. Score	Relative score	**Evaluation**	(Horos) Rel. Score	Relative score	**Evaluation**	(Medinria) Rel. Score	Relative score	**Evaluation**	(Weasis) Rel. Score	Relative score	**Evaluation**	(Ginkgo CADx) Rel. Score	Relative score	**Evaluation**
				1.00		**0.61**			**0.78**			**0.38**			**0.62**			**0.43**		
General				**0.33**		**0.62**			**0.77**			**0.38**			**0.60**			**0.41**		
General	**Technical**			0.07		0.63			0.34			0.58			0.57			0.48		
General	**Archive**			0.14		0.72			0.97			0.39			0.58			0.80		
General	**Workflow**			0.21		0.71			0.79			0.45			0.53			0.31		
General	**Visualization**			0.21		0.76			0.89			0.47			0.50			0.29		
General	**Tools**			0.21		0.63			0.89			0.15			0.86			0.48		
General	Tools	Usability		0.09		0.80		(.80)	1.00		⊤	1.00		⊤	1.00		⊤	1.00		⊤
General	Tools	Editable		0.09		0.00		⊥	1.00		⊤	0.00		⊥	1.00		⊤	0.90		(.90)
General	Tools	Transversality		0.07		0.67		(.67)	1.00		⊤	0.00		⊥	1.00		⊤	0.00		⊥
General	Tools	Undo		0.05		0.00		⊥	1.00		⊤	0.00		⊥	0.00		⊥	0.00		⊥
General	Tools	3D cursor		0.07		1.00		⊤	0.67		(.67)	0.00		⊥	1.00		⊤	0.00		⊥
General	Tools	Reference lines		0.07		0.90		(.90)	1.00		⊤	0.00		⊥	1.00		⊤	0.00		⊥
General	Tools	**Save status**		0.05		0.20			0.60			0.20			1.00			0.20		
General	Tools	**Measure**		0.09		1.00			1.00			0.23			1.00			0.85		
General	Tools	**Annotations**		0.05		0.38			1.00			0.00			0.95			0.95		
General	Tools	Annotations	Basic shapes	0.25		0.80		(.80)	1.00		⊤	0.00		⊥	1.00		⊤	0.90		(.90)
General	Tools	Annotations	Polygons	0.25		0.70		(.70)	1.00		⊤	0.00		⊥	0.80		(.80)	0.90		(.90)
General	Tools	Annotations	Arrow or marker	0.25		0.00		⊥	1.00		⊤	0.00		⊥	1.00		⊤	1.00		⊤
General	Tools	Annotations	Textual	0.25		0.00		⊥	1.00		⊤	0.00		⊥	1.00		⊤	1.00		⊤
General	Tools	Unit awareness		0.05		1.00		⊤	0.50		(.50)	0.50		(.50)	0.80		(.80)	0.70		(.70)
General	Tools	Statistics		0.07		1.00		⊤	1.00		⊤	0.00		⊥	1.00		⊤	1.00		⊤
General	Tools	SUV		0.07		1.00		⊤	0.80		(.80)	0.00		⊥	0.90		(.90)	0.00		⊥
General	Tools	Histogram		0.03		0.00		⊥	1.00		⊤	0.00		⊥	0.00		⊥	0.00		⊥
General	Tools	ROI		0.09		0.67		(.67)	1.00		⊤	0.00		⊥	0.67		(.67)	0.67		(.67)
General	Tools	Key image notes (K		0.05		0.00		⊥	0.33		(.33)	0.00		⊥	0.80		(.80)	0.00		⊥
General	**Other modalities**			0.07		0.25			0.34			0.00			0.82			0.45		
General	**Processing**			0.07		0.02			0.46			0.74			0.22			0.00		

Figure 5: Expanded evaluation results for the "tools" group and the "annotations" criterion. The user can interact to determine the information to be presented.

graphical form. The right "weights" bar shows the maximum possible score for each category, thus helping to see where each viewer is good or has room for improvement.

4. Discussion

In recent years, open-source software has emerged as a potential solution to compete with privative products in medical applications. These products have many similarities in their approach but also present great differences in provided features and functionalities. For this reason, despite the interest from the research and medical community on open-source products, the lack of clear information about the advantages and disadvantages of one product over the others, makes their use in real scenarios quite complex. Aware of this situation, the aim of this paper has been two-fold. Firstly, we have presented Starviewer, a new open-source multi-platform DICOM viewer which despite its ability to satisfy and adapt to daily clinical needs has room for some feature improvements as described below. Secondly, we have introduced an evaluation framework that reduces the complexity of selecting a viewer that can: (i) be user-configured

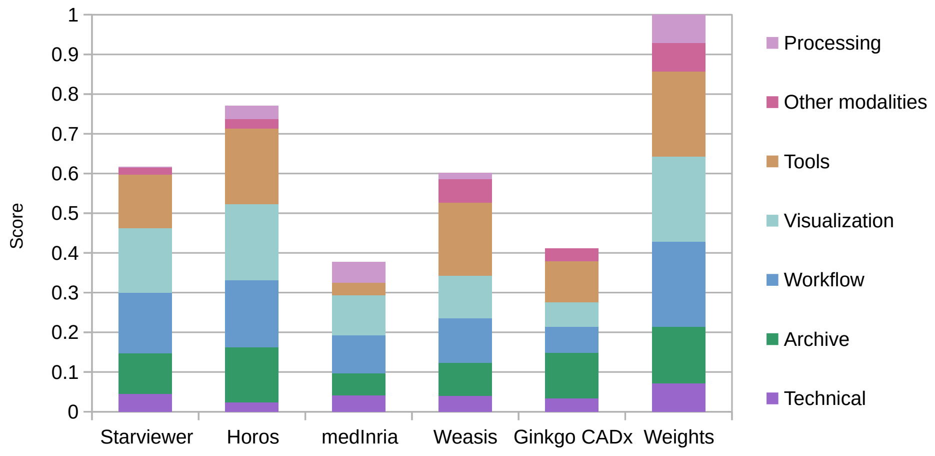


Figure 6: Graph showing the summarized results of the comparison of Starviewer with the other viewers for the general audience. The right "weights" bar shows the maximum possible score for each category.

and adapted to any user specific scenario, and (ii) be used as a complement to maturity models or quality evaluation frameworks [33] in order to perform a feature analysis.

In our case the evaluation carried out presents two main limitations. Firstly, a reduced set of viewers has been evaluated because the evaluation of each viewer in such level of detail takes a considerable effort. However, with the proposed framework it is easy to add new viewers to the comparison. Secondly, it is difficult to attain a fully objective analysis, but the hierarchical evaluation methodology and the different audiences approach forces the evaluator to perform a bottom-up analysis that helps to limit the subjectivity when setting scores, weights, and criteria. Scores were set according to the descriptions, but in some cases the assessment requires analysing and justifying the pros and cons of each candidate to set a fair final score, which may introduce some bias. Regarding weights, they were set following expert radiologists advice, but in a future work, we will consider making a survey in order to get better measurable knowledge and contemplate more specific audiences. Fortunately, all these limitations can be overcome thanks to the adaptability of this evaluation framework by the end user.

The proposed viewer has been compared using the presented evaluation framework with state of the art open-source viewers such as Ginkgo CADx, Horos, medInria, and

Weasis. Criteria have been grouped in seven categories and weighted according to user perspective. Focusing on the general audience, which takes into account all possible features, the obtained results are discussed below.

In the *technical* group, where aspects such as platform support, multi-language, and documentation are considered, Starviewer has the top score mainly due to the completeness of its documentation. Starviewer provides an extensive user and administrator documentation where all features and shortcuts are described while many of the analysed viewers do not provide it.

Regarding the *archive* category, where network protocols, local storage, and supported formats are considered, the maximum score is achieved by Horos, which supports almost all the features. The main limitations of Starviewer in this group are the lack of WADO support and its inability to convert from DICOM to non-DICOM formats and vice versa; for these reasons it is ranked in third position after Ginkgo CADx.

In the *workflow* group, where we consider features that improve user experience such as program customization, hanging protocols support, and synchronization, Horos is again the top-rated viewer mailny because it is fully customizable. Starviewer follows close in second position, being the best in synchronization and study comparison support; however it lacks a graphical hanging protocol editor.

Visualization is the group where all the 2D and 3D visualization and interaction features, and DICOM conformity are considered. Horos is first with great support for most of the criteria. Starviewer obtains a good score and is placed second, being its main drawbacks the lack of non-Latin characters support and the absence of full-screen visualization and true scaling (display of objects on screen in real size).

In the *tools* category is where we look for functionalities to measure, annotate, and comprehend datasets. Horos and Weasis obtain a very good score, with Horos slightly better due to its ability to undo actions. Starviewer is ranked third, with its main weaknesses being non-editable tools, non-savable annotations, and the lack of key image notes support.

In the *other modalities* group, where support for non-image modalities like structured reports and encapsulated documents is evaluated, the best score is achieved by Weasis thanks to its support for all special modalities. Starviewer is fourth because it only supports encapsulated documents.

Finally, in the *processing* group, which evaluates features that perform a complex analysis and usually generate new series, the top position is for medInria thanks to its extensive support for almost all the considered criteria. Starviewer obtains a bad score in this category with only some points in the segmentation criterion through the "magic ROI" tool, putting it in fourth position.

Focusing in the 2D-based audience, Weasis greatly increases its score ascending to the second position after Horos, which remains first. The main reason are the changes of the weights in the *visualization* group, where criteria that do not make sense in 2D, such as thick slab, MPR, fusion, and 3D, have their weights set to zero, thus increasing the relative weights of the other criteria.

Similarly, for the 3D-based audience, the weights of *visualization* criteria have been adjusted by increasing the ones that are more important for 3D datasets exploration. In addition the weight of the *processing* group has been slightly increased. In this case the positions stay the same as for the general audience but Weasis and Gingko CADx have significantly lower scores due to their lack of 3D support.

In our evaluation medInria has the lowest score. This is due to the fact that although it is an excellent processing tool it is not suitable for the daily clinical practice. From this evaluation we have seen that Horos is the best viewer overall, however it is only available for the macOS platform. In second position comes Starviewer, which despite having less features than Horos, is still a good viewer for daily clinical needs and has the advantage of being multi-platform. In addition, Starviewer is modular and extensible, thus new extensions and functionalities can be quickly developed according to user demands. This is a technically relevant characteristic to be taken into account, although it has not been considered in the evaluation because it is not as relevant from the end user perspective. Other viewers that have not been evaluated, like MITK [7] or 3D Slicer [4], provide specific APIs to manipulate and extend functionalities. In contrast, Starviewer extensions operate directly using VTK, reducing the degree of abstraction and enhancing VTK-based code portability.

From the aforementioned analysis, one can conclude that Starviewer main advantages are technical documentation, productive workflow, and extensibility. Main disadvantages are lack of WADO support, DICOM conversions, lack of hanging protocol editor, Unicode character support, non-editable tools, key image notes and lack of special modalities like EKG. All those shortcomings will be addressed in our future developments.

We are conscious that medical imaging software paradigm is changing to software-as-a-service where solutions run in the cloud. However there are still certain technological issues that need further development to provide the advantages of desktop applications [53]. On the one hand, implementing certain features such as interactive oblique reconstructions and 3D volume rendering on the client side require that all the images are available in the client memory and that requires powerful enough hardware to perform such operations, thus negating the advantage of requiring only cheap thin clients. On the other hand, if these and other features are implemented on the server side, the viewer is highly dependent on and limited by the network bandwidth, latency and stability. In addition, a high client concurrency can lead to peformance degradation in the server [54]. For these reasons, we consider that Starviewer can be of interest to the health community.

Summary points

What was already known on the topic:

- DICOM viewers are essential in many clinical processes, either to diagnose, plan operations, or follow up the evolution of pathologies.
- The interest in open-source DICOM viewers has increased considerably.
- Selecting the DICOM viewer that better fits user needs can be a complex task.

What this study added to our knowledge:

- A new open-source multi-platform DICOM viewer that supports main needs of image diagnosis experts and the creation of custom extensions for specific workflows.

- An adaptable evaluation framework that considers different audiences and criteria weighted according to user needs and that can be used as a complement to maturity models and quality evaluation frameworks.
- An application example where Starviewer and state-of-the art open-source DICOM viewers are compared.

Informed consent

All anonymous subjects depicted in this paper signed an informed consent form allowing the use of the images for research purposes.

Authors' contributions

All authors have contributed equally.

Acknowledgements

We want to thank all developers who contributed to Starviewer as well as others which have contributed to the project in some manner. We also want to thank Institut de Diagnòstic per la Imatge (IDI), our partner since 2004. This work has been funded in part by grants from the Spanish Government (Nr. TIN2016-75866-C3-3-R) and from the Catalan Government (Nr. 2017-SGR-1101).

Declarations of interest

Declarations of interest: none.

References

[1] National Electrical Manufacturers Association (NEMA), Digital Imaging and Communications in Medicine (DICOM), [Internet] (2018) [cited 08/01/2018]. URL `https://www.dicomstandard.org/current/`

[2] D. H. K. Huang, PACS and Imaging Informatics: Basic Principles and Applications, 2nd Edition, Wiley-Blackwell.

[3] G. Valeri, F. A. Mazza, S. Maggi, D. Aramini, L. La Riccia, G. Mazzoni, A. Giovagnoni, Open source software in a practical approach for post processing of radiologic images, La radiologia medica 120 (3) (2015) 309–323 (Mar 2015). `doi:10.1007/s11547-014-0437-5`.

[4] A. Fedorov, R. Beichel, J. Kalpathy-Cramer, J. Finet, J.-C. Fillion-Robin, S. Pujol, C. Bauer, D. Jennings, F. Fennessy, M. Sonka, J. Buatti, S. Aylward, J. V. Miller, S. Pieper, R. Kikinis, 3D Slicer as an image computing platform for the Quantitative Imaging Network, Magnetic Resonance Imaging 30 (9) (2012) 1323–1341 (2012). `doi:10.1016/j.mri.2012.05.001`.

[5] 3D Slicer community, 3DSlicer, [Internet] (2018) [cited 18/01/2018].
URL `https://www.slicer.org`

[6] Inria, medInria, [Internet] (2018) [cited 16/01/2018].
URL `http://med.inria.fr`

[7] I. Wolf, M. Vetter, I. Wegner, T. Böttger, M. Nolden, M. Schöbinger, M. Hastenteufel, T. Kunert, H.-P. Meinzer, The Medical Imaging Interaction Toolkit, Medical Image Analysis 9 (6) (2005) 594–604 (2005). `doi:10.1016/j.media.2005.04.005`.

[8] German Cancer Research Center, Division of Medical Image Computing, The Medical Imaging Interaction Toolkit (MITK), [Internet] (2018) [cited 18/01/2018].
URL `http://mitk.org/wiki/MITK`

[9] Kitware Inc., VolView, [Internet] (2018) [cited 16/01/2018].
URL `https://www.kitware.com/volview/`

[10] IRCAD, VR-Render, [Internet] (2014) [cited 22/08/2014 (archived)].
URL `https://web.archive.org/web/20140822230414/http://www.ircad.fr:80/recherche/rd/rd.php`

[11] O. Ratib, A. Rosset, Open-source software in medical imaging: development of OsiriX, International Journal of Computer Assisted Radiology and Surgery 1 (4) (2006) 187–196 (Dec 2006). doi:10.1007/s11548-006-0056-2.

[12] Pixmeo SARL, OsiriX, [Internet] (2018) [cited 18/01/2018].
URL https://www.osirix-viewer.com/osirix/overview

[13] D. Haak, C.-E. Page, T. M. Deserno, A Survey of DICOM Viewer Software to Integrate Clinical Research and Medical Imaging, Journal of Digital Imaging 29 (2) (2016) 206–215 (Apr 2016). doi:10.1007/s10278-015-9833-1.

[14] M. Mcauliffe, F. Lalonde, D. McGarry, W. Gandler, K. Csaky, B. Trus, Medical Image Processing, Analysis & Visualization in Clinical Research, in: Proceedings of the 14th IEEE Symposium on Computer-Based Medical Systems, Vol. 14, 2001, pp. 381–386 (02 2001).

[15] National Institutes of Health, MIPAV (Medical Image Processing, Analysis, and Visualization), [Internet] (2018) [cited 08/01/2018].
URL https://mipav.cit.nih.gov/

[16] dcm4che, Weasis, [Internet] (2018) [cited 08/01/2018].
URL https://nroduit.github.io/en/basics/architecture/

[17] The Horos Project, Horos, [Internet] (2018) [cited 26/07/2018].
URL https://horosproject.org/

[18] T. Mettler, P. Rohner, Situational maturity models as instrumental artifacts for organizational design, http://www.alexandria.unisg.ch/Publikationen/67758 (01 2009). doi:10.1145/1555619.1555649.

[19] A. Rocha, Evolution of information systems and technologies maturity in healthcare, International journal of healthcare information systems and informatics 6 (2011) 28–36 (04 2011). doi:10.4018/jhisi.2011040103.

[20] R. van de Wetering, R. Batenburg, A pacs maturity model: A systematic meta-analytic review on maturation and evolvability of pacs in the hospital enterprise,

International journal of medical informatics 78 (2008) 127–40 (09 2008). `doi:10.1016/j.ijmedinf.2008.06.010`.

[21] J. Carvalho, A. Rocha, A. Abreu, Maturity models of healthcare information systems and technologies: a literature review, Journal of Medical Systems 40 (2016) 10 (04 2016). `doi:10.1007/s10916-016-0486-5`.

[22] U. e Laila, A. Zahoor, K. Mehboob, S. Natha, Comparison of open source maturity models, Procedia Computer Science 111 (2017) 348–354, the 8th International Conference on Advances in Information Technology (2017). `doi:10.1016/j.procs.2017.06.033`.
URL `http://www.sciencedirect.com/science/article/pii/S1877050917312061`

[23] M. A. Babar, L. Zhu, R. Jeffery, A Framework for Classifying and Comparing Software Architecture Evaluation, in: In: Proceedings Australian Software Engineering Conference (ASWEC). (2004, 2004, pp. 309–318 (2004).

[24] M.-P. Gagnon, M. Desmartis, M. Labrecque, J. Car, C. Pagliari, P. Pluye, P. Fremont, J. Gagnon, N. Tremblay, F. Légaré, Systematic review of factors influencing the adoption of information and communication technologies by healthcare professionals, Journal of medical systems 36 (2012) 241–77 (02 2012). `doi:10.1007/s10916-010-9473-4`.

[25] P. Nagy, Open Source in Imaging Informatics, Journal of Digital Imaging 20 (1) (2007) 1–10 (Nov 2007). `doi:10.1007/s10278-007-9056-1`.
URL `https://doi.org/10.1007/s10278-007-9056-1`

[26] G. L. Presti, M. Carbone, D. Ciriaci, D. Aramini, M. Ferrari, V. Ferrari, Assessment of DICOM Viewers Capable of Loading Patient-specific 3D Models Obtained by Different Segmentation Platforms in the Operating Room, Journal of Digital Imaging 28 (5) (2015) 518–527 (Oct 2015). `doi:10.1007/s10278-015-9786-4`.
URL `https://doi.org/10.1007/s10278-015-9786-4`

[27] F. Kamseu, N. Habra, Adoption of open source software: Is it the matter of quality? (01 2004).

[28] Raphaël Semeteys, Qualification and Selection of Opensource Software, [Internet] (2018) [cited 18/12/2019].
URL `https://www.qsos.org/`

[29] V. del Bianco, L. Lavazza, S. Morasca, D. Taibi, D. Tosi, The qualispo approach to oss product quality evaluation, 2010 (05 2010). `doi:10.1145/1833272.1833277`.

[30] Qualipso, Qualification and Selection of Opensource Software, [Internet] (2013) [cited 13/16/2013].
URL `https://web.archive.org/web/20130613043131/http://www.qualipso.eu/`

[31] W. J. Sung, J. H. Kim, S. Y. Rhew, A quality model for open source software selection, in: Sixth International Conference on Advanced Language Processing and Web Information Technology (ALPIT 2007), 2007, pp. 515–519 (Aug 2007). `doi:10.1109/ALPIT.2007.81`.

[32] L. Aversano, M. Tortorella, Evaluating the quality of free/open source systems: A case study, Vol. 73, 2010, pp. 119–134 (06 2010). `doi:10.1007/978-3-642-19802-1_9`.

[33] L. Aversano, M. Tortorella, Quality evaluation of floss projects: Application to erp systems, Information and Software Technology 55 (2013) 1260–1276 (07 2013). `doi:10.1016/j.infsof.2013.01.007`.

[34] The Qt Company, Qt project, [Internet] (2017) [cited 19/11/2017].
URL `https://www.qt.io/`

[35] Kitware Inc., The Visualization ToolKit, [Internet] (2017) [cited 19/11/2017].
URL `https://www.vtk.org`

[36] Kitware Inc., The Insight Segmentation and Registration ToolKit, [Internet] (2017) [cited 19/11/2017].
URL `https://www.itk.org`

[37] OFFIS e.V., DCMTK ToolKit, [Internet] (2017) [cited 19/11/2017].
URL `http://dcmtk.org/`

[38] M. Malaterre, Grassroots DICOM, [Internet] (2018) [cited 03/01/2018].
URL `https://gdcm.sourceforge.net/`

[39] KDE e.V, ThreadWeaver, [Internet] (2018) [cited 03/01/2018].
URL `https://download.kde.org/Attic/frameworks/5.3.0/threadweaver-5.3.0.tar.xz`

[40] Google, Breakpad, [Internet] (2018) [cited 03/01/2018].
URL `https://chromium.googlesource.com/breakpad/breakpad/`

[41] S. Graf, B. List, An Evaluation of Open Source E-Learning Platforms Stressing Adaptation Issues, in: Proceedings of the Fifth IEEE International Conference on Advanced Learning Technologies, ICALT '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 163–165 (2005). `doi:10.1109/ICALT.2005.54`.
URL `https://doi.org/10.1109/ICALT.2005.54`

[42] N. Cavus, The Evaluation of Learning Management Systems Using an Artificial Intelligence Fuzzy Logic Algorithm, Adv. Eng. Softw. 41 (2) (2010) 248–254 (Feb. 2010). `doi:10.1016/j.advengsoft.2009.07.009`.
URL `http://dx.doi.org/10.1016/j.advengsoft.2009.07.009`

[43] A. Crabb, I Do Imaging, [Internet] (2018) [cited 31/07/2018].
URL `https://idoimaging.com/`

[44] AlternativeTo, AlternativeTo - Crowdsourced software recommendations, [Internet] (2018) [cited 31/07/2018].
URL `https://alternativeto.net/`

[45] GitHub, GitHub, [Internet] (2018) [cited 31/07/2018].
URL `https://github.com/`

[46] Atlassian Pty Ltd, Bitbucket, [Internet] (2018) [cited 31/07/2018].
URL `https://bitbucket.org/`

[47] GitLab, GitLab, [Internet] (2018) [cited 31/07/2018].
URL `https://gitlab.com/`

[48] Slashdot Media, SourceForge - Download, Develop and Publish Free Open Source Software, [Internet] (2018) [cited 31/07/2018].
URL `https://sourceforge.net/`

[49] Aliza Medical Imaging, Aliza Medical Imaging & DICOM Viewer, [Internet] (2018) [cited 26/07/2018].
URL `http://www.aliza-dicom-viewer.com`

[50] Amide, AMIDE: Amide's a Medical Imaging Data Examiner, [Internet] (2012) [cited 26/07/2018].
URL `http://http://amide.sourceforge.net/`

[51] G. Wollny, Ginkgo CADx, [Internet] (2018) [cited 08/01/2018].
URL `http://ginkgo-cadx.com/en/`

[52] Centro de Tecnologia da Informação Renato Archer, InVesalius, [Internet] (2018) [cited 26/07/2018].
URL `https://www.cti.gov.br/en/invesalius`

[53] S. Min, Z. Wang, N. Liu, An evaluation of html5 and webgl for medical imaging applications, Journal of Healthcare Engineering 2018 (2018) 1–11 (08 2018). `doi:10.1155/2018/1592821`.

[54] R. Yuan, M. Luo, Z. Sun, S. Shi, P. Xiao, Q. Xie, Rayplus: a web-based platform for medical image processing, Journal of Digital Imaging 30 (11 2016). `doi:10.1007/s10278-016-9920-y`.

Appendix A. Workflow to create a Starviewer extension

Focusing on final users, the main advantage of Starviewer is the possibility to create custom extensions for specific workflows. Currently, Starviewer only supports extensions as static libraries, thus one needs to be able to build Starviewer from source. Once the build environment is set up, creating an extension is very simple as illustrated in the following example.

Extensions must be placed under the `src/extensions` directory, and there are three possible directories to choose from: (i) `main`, intended for official stable extensions; (ii) `contrib`, intended for stable third-party contributed extensions; and (iii) `playground`, intended for unstable or experimental extensions. Thus, the example extension is a nice fit for `playground`.

The workflow to create an extension can be summarized in four steps: (i) create the extension subproject; (ii) create the bare minimum classes, including the UI; (iii) generate the translation files and add them to a resource file; and (iv) implement the actual functionality for the extension. Now we will extend on each of these steps.

Note that in this section we will include only a few code snippets which have been simplified for the sake of readability, omitting include guards, include directives, namespace declarations, and safety checks, and occasionally merging header and source files. Full code listings can be found in the appendix and in the supporting material.

A.1. Extension subproject

The first step is creating a subproject for the extension. We have to create a directory called `example` under `src/extensions/playground`. Inside this directory we must create a `qmake` project file named `example.pro` with this content:

```
EXTENSION_DIR = $$PWD
include(../../basicconfextensions.pri)
```

Finally, we have to add a reference to this new subproject in parent projects. Specifically, we must add `example` to the `SUBDIRS` variable in `src/extensions/playground/playground.pro` and the `PLAYGROUND_EXTENSIONS` variable in `src/extensions.pri`.

A.2. Minimum code

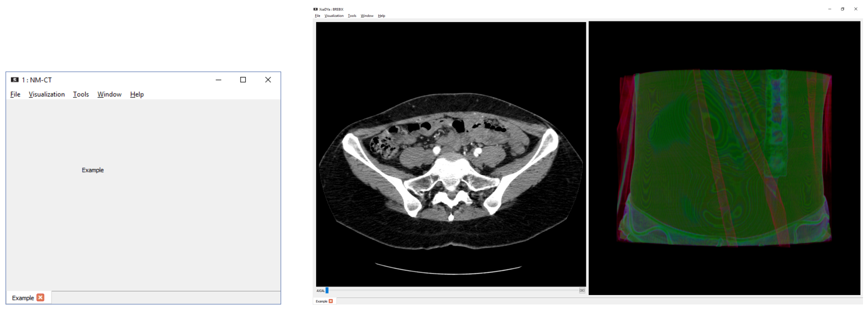
The bare minimum code required for an extension consists of the main extension class and a special class that is used by the core to initialize the extension. All new files from now on will be placed inside the `example` directory. By convention, all UI classes start with a 'Q' and all the code is placed in the `udg` namespace.

First, we create an UI file named `qexampleextensionbase.ui` which, for now, is just a `QWidget` containing a `QLabel` with the text "Example". We name the main widget `"QExampleExtensionBase"`, and may be created using the Qt Designer graphical editor or the integrated one inside Qt Creator.

Then we create the main extension class named `QExampleExtension` which inherits `QWidget` publicly and `Ui::QExampleExtensionBase` privately. The latter is generated by `uic`, the Qt UI compiler, from the previously created `.ui` file. This class has just a constructor that calls `setupUi(this)`.

Now, we need to create a special class named `ExampleExtensionMediator`. It is a subclass of `ExtensionMediator` and must implement two methods to initialize the extension and to give it a unique ID. In the header file, we also need to create a static instance of `InstallExtension` to make the extension known to Starviewer. This is all done in the following snippet:

```
class ExampleExtensionMediator : public ExtensionMediator {
    Q_OBJECT
public:
    explicit ExampleExtensionMediator(QObject *parent = nullptr) {}
    bool initializeExtension(QWidget *extension,
                            const ExtensionContext &extensionContext) override {
        QExampleExtension *exampleExtension;
        if (!(exampleExtension = qobject_cast<QExampleExtension*>(extension))) {
            return false;
        }
        return true;
    }
    DisplayableID getExtensionID() const override {
        return DisplayableID("ExampleExtension", tr("Example"));
    }
};
static InstallExtension<QExampleExtension, ExampleExtensionMediator>
```



(a) (b)

Figure A.1: Screenshots of the Example extension (a) containing just a label and (b) having a 2D and a 3D viewer.

```
    registerExampleExtension;
```

Finally, we add both classes and the UI file to the project file `example.pro`.

A.3. Translations

Translation files must be generated and added to the project, even if they are empty. To generate these files we have to execute `lupdate example.pro` from the `example` directory; this will create a few `.ts` files. Then, we create a resource file `example.qrc` that includes the compiled translations (which will be created during the build from the `.ts` files, with the same names but with the `.qm` extension) under the prefix `"/extensions/ExampleExtension"`. The last part of the prefix must match the ID given in the extension mediator. Finally, we add the resource file to the `RESOURCES` variable in `example.pro`.

After this step, we can build Starviewer and try the new extension, as illustrated at Fig. A.1(a). It can be launched by selecting *Example* in the *Visualization* menu after opening a study.

A.4. Adding functionality

Having a minimal extension working, the next step is adding actual functionality to it. In this example we will add a 2D and a 3D viewer. Viewers can be added in the graphical editor as `QWidgets` and then promoted to `Q2DViewerWidget` and `Q3DViewer`, respectively.

Then we add a `setPatient` method to `QExampleExtension` with the following code:

```
void QExampleExtension::setPatient(Patient *patient) {
    m_2DViewer->setInputAsynchronously(patient->getVolumesList().first());
    m_3DViewer->setInput(patient->getVolumesList().first());
}
```

Finally we call this method from `ExampleExtensionMediator`, getting the patient from the `ExtensionContext`:

```
    exampleExtension->setPatient(extensionContext.getPatient());
```

If we build and launch the extension in this moment, we can already see both viewers showing the first series of the opened study, as seen in Fig. A.1(b). Both viewers allow to independently change the series using the right click menu. The 2D viewer allows changing slices with a slider, but the 3D viewer does not support any interactions yet.

To add interactions to the viewers, we need to register and activate several tools in each one. This is done with the help of a `ToolManager`. Since each viewer will have a different set of tools, we will need separated `ToolManager` instances for each one. We add the new code to the extension's constructor:

```
QExampleExtension::QExampleExtension(QWidget *parent) : QWidget(parent) {
    setupUi(this);

    ToolManager *toolManager2D = new ToolManager(this);
    toolManager2D->registerTool("ZoomTool");            // left button
    toolManager2D->registerTool("TranslateTool");       // middle button
    toolManager2D->registerTool("WindowLevelTool");     // right button
    toolManager2D->registerTool("SlicingKeyboardTool"); // keyboard
    toolManager2D->registerTool("SlicingWheelTool");    // wheel
    toolManager2D->setupRegisteredTools(m_2DViewer->getViewer());
    toolManager2D->triggerTools({"ZoomTool", "TranslateTool", "WindowLevelTool",
                                 "SlicingKeyboardTool", "SlicingWheelTool"});

    ToolManager *toolManager3D = new ToolManager(this);
    toolManager3D->registerTool("ZoomTool");        // left button
    toolManager3D->registerTool("TranslateTool");   // middle button
    toolManager3D->registerTool("Rotate3DTool");    // right button
    toolManager3D->setupRegisteredTools(m_3DViewer);
    toolManager3D->triggerTools({"ZoomTool", "TranslateTool", "Rotate3DTool"});
}
```

If we rebuild and launch the extension again, we can try the new features. In the 2D viewer we can zoom while holding the left button, pan while holding the middle button, adjust the window level or VOI LUT while holding the right button, and change slices with the keyboard arrows and the mouse wheel. In the 3D viewer we can zoom while holding the left button, pan while holding the middle button, and rotate the image while holding the right button.

Appendix B. Score calculation

The motivation of this appendix is to explain with detail how the actual scores are calculated following the reduced but complete example of Fig. 2. Note that the actual spreadsheet has many more (hidden) intermediate columns that propagate values up and down in order to accomplish the tree-like structure calculations.

First of all we have to define some concepts: (i) *nodes*, which are rows, and must have a weight; (ii) *leaf nodes*, which are rows without children nodes, are the only directly evaluated items for each analysed software, and their scores always correspond with the evaluation; and (iii) *container nodes*, which have children nodes, are never directly evaluated, and their score is the weighted sum of their direct children scores.

The columns in bold are the ones to be considered user editable. Almost all numeric columns have values ranging from 0 to 1; being an exception the *weights* editable column, that may take any positive number. Then, for each sibling, all weights are summed and normalized to a 0 to 1 range that is placed in the *relative norm. w.* column. The sum of *relative norm. w.* across all siblings is exactly 1. The reason behind this methodology is to make the weight setting process more easy and natural.

Container nodes scores are computed recursively, starting from leaf nodes and ascending up to the root node, according to the following process: (i) each *rel. score* of each analysed software is multiplied by the *relative norm. w.*; (ii) each *rel. score* of each direct child is summed; and (iii) the result of the summation is the *rel. score* that the *container node* takes.